Fast Probabilistic File Fingerprinting for Big Data

Supplementary Text

Konstantin Tretyakov, Sven Laur, Geert Smant,
Jaak Vilo and Pjotr Prins

1 Estimation of δ-variability

1.1 Data

In order to estimate δ-variability we obtained several biological datasets, including DNA sequence data, DNA annotation data, Affymetrix microarray chip data and RNASeq raw data. Each dataset was originally stored in the warehouse in the form of gzip-compressed files. For each compressed dataset we created an "uncompressed" version by unpacking all the files in the dataset. Dataset descriptions are provided in Table 1, dataset statistics are listed in Table 2. Detailed dataset listings are provided in the Supplementary Files, in the `datasets` subdirectory.

1.2 δ-variability estimation

Recall that the δ-variability for a pairs is defined as the proportion of pairwise different blocks. Albeit the definition provided in the paper assumes files of equal size, for the purposes of the experiment, we extend it to arbitrary file pairs. Given two files of different sizes we assume the shorter file to be padded by "NA"-bytes to the length of the longer file (the "NA"-bytes are treated as different from any other byte value). δ-variability of the two files is then simply the proportion of pairwise different bytes in the now equal-length files.

The δ-variability for a dataset is defined as the smallest δ-variability of a pair of files over all file pairs. Due to the sizes of the data involved, a brute-force enumeration of all pairs of files followed by δ-variability computation is unreasonably slow for some of the datasets. Consequently, we used approximations to detect most similar file pairs. The code used in our experiments is provided in the Supplementary Files, in the `variability` subdirectory.

1.2.1 Smaller datasets

For smaller datasets we first list all pairs of files with relative size difference no more than 30%. We then compute an approximation to δ-variability by only

comparing a single kilobyte from each megabyte of the file. We sort all pairs by this approximate value and perform exact comparison on the best 1000 pairs.

1.2.2 Larger datasets

For larger file collections (the GPL570/* and E61/* datasets) the approach above is still unreasonably slow, so we use a faster approximation. In that, we first compute a fingerprint for each file using Algorithm 1.

Algorithm 1 APPROXIMATE-COMPARISON-FINGERPRINT

Require: File F of size s bytes.

procedure COMPUTE-FINGERPRINT
    sclass $\leftarrow$ round(ln(s))
    return Extract-Bytes(sclass) + Extract-Bytes(sclass $-$ 1)
end procedure

function EXTRACT-BYTES(sclass)
    lastbyte $\leftarrow$ floor(exp(sclass $-$ 0.6))
    step $\leftarrow$ floor(lastbyte/20)
    return $\{$(sclass, $i \cdot$ step, $F[i \cdot \text{step}]$) : $i \in 1 \ldots 20\}$
end function

The idea behind the fingerprint computation is the following. For each file we first determine its "size class", which is simply the logarithm of file size, rounded to the closest integer. We then fix the "last sampling byte" for a file of a given size class as

$$\text{lastbyte} = \exp(\text{sclass} - 0.6).$$

The rationale for this choice is simply to pick a byte which would be guaranteed to be within the span of any file of a given size class and located somewhat towards the end of the file. Note that the smallest possible size for a given size class is exp(sclass $-$ 0.5).

Finally, we read 20 bytes spaced at regular intervals between byte 0 and lastbyte. This choice guarantees that for all files of the same size class we read 20 bytes at exactly the same positions. In addition, for each file we also sample the 20 bytes as if it belonged to a size class (sclass $-$ 1). As a result, from each file we sample 40 bytes, so that for any two files of the same size class all 40 are sampled at exactly the same positions, and for two files of neighboring size classes, 20 of them are sampled at the same positions.

Having computed the fingerprints we can efficiently find file pairs which have at least four similar bytes in their fingerprints. We use those as the candidate pairs for exact evaluation.

To ensure reliability of the results, we repeated candidate detection using several slightly different settings of the described algorithm.

1.2.3 Filtering equivalent files

The GPL570/* group datasets contained numerous file pairs, which were equivalent in one of the following senses:

- Files have different names, but are byte-wise equal.
- Files are compressed versions of byte-wise equal files.
- One of the two files is a prefix of the other.
- One of the two files, when unpacked, is a prefix of the other file, when unpacked.

We filtered those pairs from our comparison as being essentially equal or irrelevant for purposes of PFFF fingeprinting assessment.

Dataset	**Description**
E61/dat	Ensembl v61 genome annotation (DAT) and DNA sequence (FASTA) files in both compressed (gzip) and uncompressed forms.
E61/dat.gz	
E61/fa	
E61/fa.gz	
GPL570/cel	Microarray files for the HG U133 Plus chip from GEO (all files of GPL570 platform as of 03.2011). Affymetrix CEL and CHP format files, in compressed (gzip) and uncompressed form.
GPL570 /cel.gz	
GPL570/chp	
GPL570 /chp.gz	
BioC2.7/ BSGenome	Raw DNA sequence from the Bioconductor package BSGenome, in compressed and uncompressed forms.
BioC2.7/ BSGenome/u	
YaleTFBS/ bedGraph4	Raw ChIP-seq data from the YaleTFBS dataset of the ENCODE project. Four different file types, both in compressed and uncompressed forms.
YaleTFBS/ bedGraph4.gz	
YaleTFBS/ fastq	
YaleTFBS/ fastq.gz	
YaleTFBS/ tagAlign	
YaleTFBS/ tagAlign.gz	
YaleTFBS/ wig	
YaleTFBS/ wig.gz	

Table 1: Datasets used to measure δ-variability.

Dataset	**Number of files**	**Total size (in GB)**	**File size (in MB)**		δ
			Min	**Max**	
E61/dat	5544	169.57	5.04	1385.14	0.782
E61/dat.gz	5544	42.92	1.02	400.21	0.996
E61/fa	1484	498.51	3.47	13306.96	0.015
E61/fa.gz	1484	95.25	1.0	973.15	0.594
GPL570/cel	59892	1022.29	1.92	173.27	0.000
GPL570 /cel.gz	59892	330.09	1.13	48.84	0.000
GPL570/chp	2535	63.30	1.67	36.50	0.209
GPL570 /chp.gz	2535	26.36	1.02	23.05	0.995
BioC2.7/ BSGenome	513	8.45	1.00	117.17	0.981
BioC2.7/ BSGenome/u	513	32.41	1.62	447.40	0.000
YaleTFBS/ bedGraph4	171	139.91	216.73	2447.62	0.924
YaleTFBS/ bedGraph4.gz	171	31.45	52.89	551.80	0.996
YaleTFBS/ fastq	388	541.99	199.25	4469.89	0.919
YaleTFBS/ fastq.gz	388	160.75	49.55	1564.84	0.996
YaleTFBS/ tagAlign	520	279.45	79.95	2357.32	0.544
YaleTFBS/ tagAlign.gz	520	96.70	27.86	815.63	0.994
YaleTFBS/ wig	33	10.66	188.92	693.66	0.912
YaleTFBS/ wig.gz	33	3.27	59.76	207.93	0.996

Table 2: Metrics of the datasets used in variability estimation.

2 PFFF Performance Evaluation

We have implemented PFFF as a command-line tool (available at `http://biit.cs.ut.ee/pfff` for Windows and Linux). We used our implementation to compare hashing performance to MD5 in several usage contexts. Table 3 lists the experimental settings we considered.

Test	Description
HTTP	In order to compute file hash, a HTTP connection is made from a researcher's laptop to a server in the same university network via HTTP connection. Files on the server are stored on a 7.2krpm hard disk attached via SCSI.
NFS	Hash is computed for files residing on a volume mounted over NFS. The mounted volume is, a heavily-used university-wide storage area network (SAN) disk array (*IBM DS 3400 FC*).
USB	Hash is computed for files residing on a portable 5.4krpm hard drive (*MyDrive Platinum 500GB*) attached via USB-to-SATA converter box.
SSD	An SSD hard drive (*HP Compaq 128 GB*), via SATA.
Flash	A no-name 2GB USB-stick SSD (flash) drive.
RAID1	A RAID1 (2 x 10krpm SAS hard drive) disk array.

Table 3: Experimental settings used to assess PFFF performance.

Note that the main text of the paper does not present the last two experiments as those are largely similar to the SSD and USB cases.

For NFS, USB, SSD, Flash and RAID1 experiments we compared the timings of the following two commands:

```
pfff -k 1 -n <sample_size> <file_name> > /dev/null
md5sum <file_name> > /dev/null
```

For HTTP we compared the timing of the following two commands:

```
pfff -k 1 -n <sample_size> -W <host> <file_name> > /dev/null
curl -s <file_url> | md5sum > /dev/null
```

We tested PFFF with two different values for the `<sample_size>` parameter: 32 and 325, corresponding to what we assessed to be the suitable for compressed datasets and "worst case" scenarios.

In each experiment we first generate several hundred files with random data and randomly chosen sizes in the appropriate size range. After that we ensure the intermediate caches of the newly created files have been flushed (e.g. by unplugging the device in the case of USB, using `/proc/sys/vm/drop_caches`, writing sufficiently large files or simply waiting long enough). Finally, we hash one third of the files using PFFF-32, one third using PFFF-325, and the remaining third using MD5.

The results are presented in Figures 1-6. The scripts used to perform the experiments as well as raw experimental data are available in the Supplementary Files, in the `performance` subdirectory.

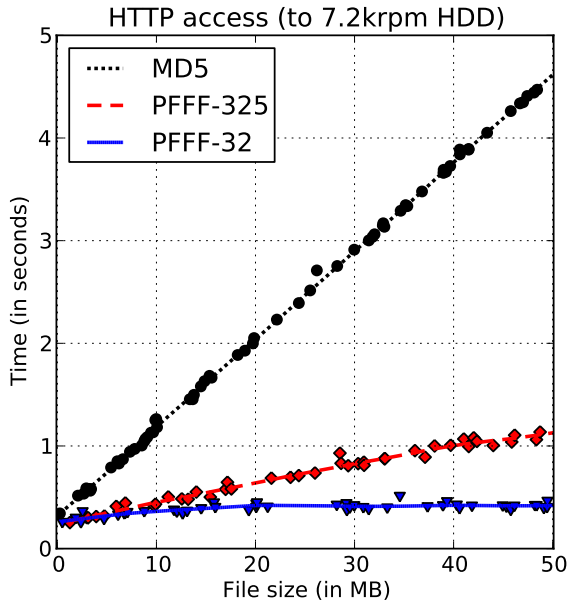

Figure 1: HTTP access performance.

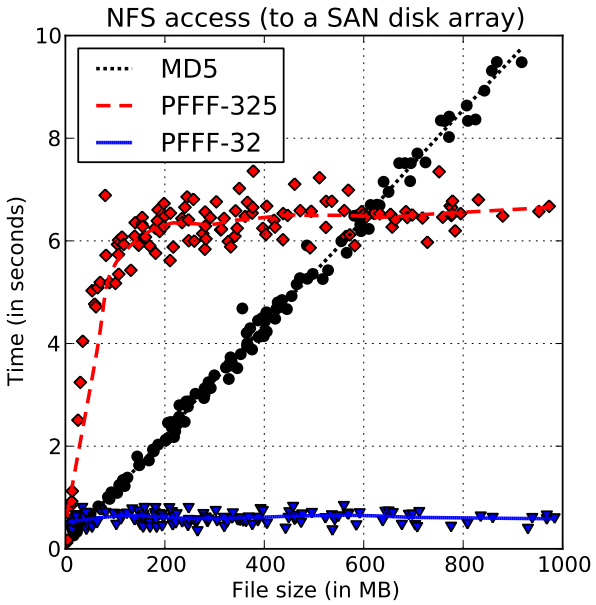

Figure 2: NFS access performance.

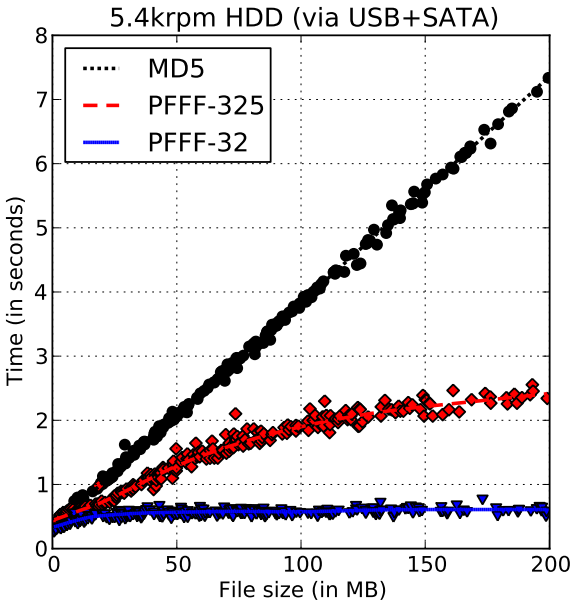

Figure 3: 5.4krpm HDD access performance.

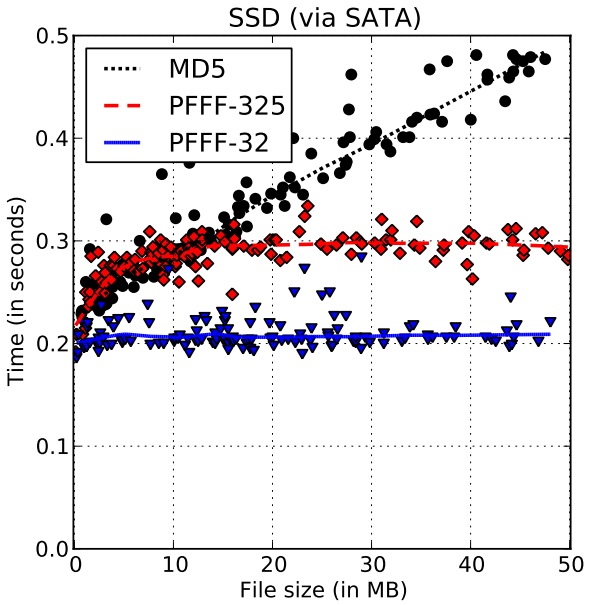

Figure 4: SSD access performance.

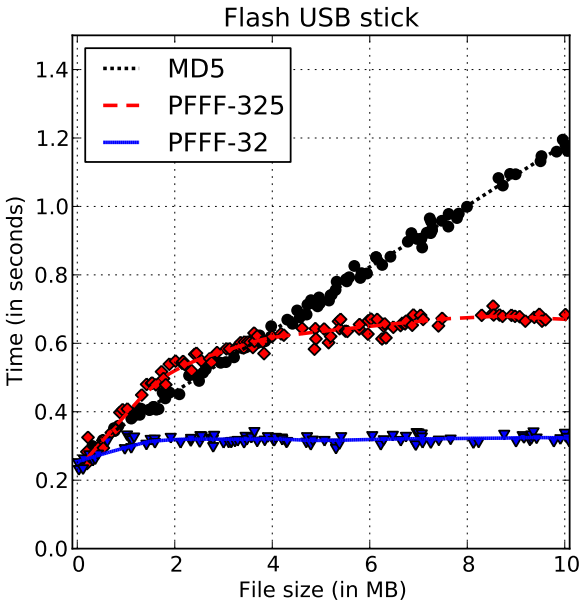

Figure 5: USB flash stick access performance.

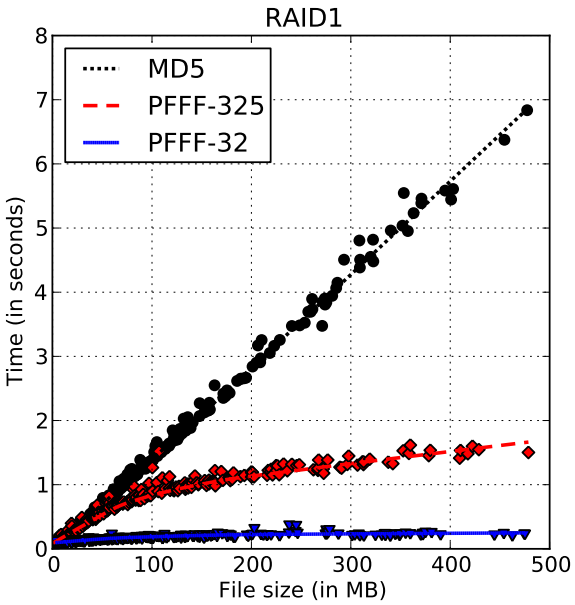

Figure 6: RAID1 2-disk array access performance.

3 Application to Duplicate Detection

In order to find redundant files in the GPL570/cel.gz dataset we applied PFFF with sample size $\ell = 11$ to compute hashes for all files and regarded all hash collisions as instances of redundant files. We repeated the experiment with three different values of the randomization key (1,2,3) and results were exactly the same all three times. Manual examination of some of the discovered file clusters confirmed that those files indeed contain the same data. Note that this could not have been detected by plain hashing, because the data files are only equivalent after unpacking – their gzip-compressed versions differ slightly in the gzip file header.

The scripts used to perform the experiment are available in the Supplementary Files, in the `duplicates` subdirectory.